Electronics **2015**, *4*, 94-117; doi:10.3390/electronics4010094

OPEN ACCESS

electronics

ISSN 2079-9292

www.mdpi.com/journal/electronics

Article

A Geometric Algebra Co-Processor for Color Edge Detection

Biswajit Mishra [1], Peter Wilson [2,*] and Reuben Wilcock [2]

[1] VLSI & Embedded Research Lab, DA-IICT, Gandhinagar 382 007, India; E-Mail: biswajit_mishra@daiict.ac.in

[2] Electronics and Computer Science, University of Southampton, Highfield Campus, Southampton SO17 1BJ, UK; E-Mail: rw3@ecs.soton.ac.uk

* Author to whom correspondence should be addressed; E-Mail: prw@ecs.soton.ac.uk; Tel.: +44-23-8059-4162; Fax: +44-23-8059-2901.

Academic Editor: Ignacio Bravo-Muñoz

Received: 19 November 2014 / Accepted: 7 January 2015 / Published: 26 January 2015

Abstract: This paper describes advancement in color edge detection, using a dedicated Geometric Algebra (GA) co-processor implemented on an Application Specific Integrated Circuit (ASIC). GA provides a rich set of geometric operations, giving the advantage that many signal and image processing operations become straightforward and the algorithms intuitive to design. The use of GA allows images to be represented with the three R, G, B color channels defined as a single entity, rather than separate quantities. A novel custom ASIC is proposed and fabricated that directly targets GA operations and results in significant performance improvement for color edge detection. Use of the hardware described in this paper also shows that the convolution operation with the rotor masks within GA belongs to a class of linear vector filters and can be applied to image or speech signals. The contribution of the proposed approach has been demonstrated by implementing three different types of edge detection schemes on the proposed hardware. The overall performance gains using the proposed GA Co-Processor over existing software approaches are more than 3.2× faster than GAIGEN and more than 2800× faster than GABLE. The performance of the fabricated GA co-processor is approximately an order of magnitude faster than previously published results for hardware implementations.

Keywords: geometric algebra; color edge detection; geometric algebra hardware

1. Introduction

With the pervasive nature of computing devices that use increasingly complex video processing, there is becoming an ever greater need for faster and more efficient processing of image and video data. One of the key areas in this field is edge detection, particularly in color images, and while it is possible to carry out image processing using software, often it is simply too slow. In order to address this key issue of performance we have designed and implemented a Geometric Algebra Co-Processor that can be applied to image processing, in particular edge detection.

Edge detection is one of the most basic operations in image processing and can be applied to both gray scale and color images. For gray scale images, edges are defined as the discontinuity in the brightness function, whereas in color images they are defined as discontinuities along adjacent regions in the RGB color space. Traditional color edge detection involves applying the uncorrelated monochrome or scalar based technique to three correlated color channels. However, to smooth along a particular color component within the image, component-wise filtering gives incorrect results as described by [1–3]. Different techniques exists that treat color as a 3-D vector to avoid such problems [4,5]. The techniques of convolution and correlation are quite common to image processing algorithms for scalar fields. Standard techniques used to identify the edges and critical features of an image use the rotational and curvature properties of vector fields, which is increasingly a popular method [6]. A combination of the scalar and vector field techniques has been extended to vector fields for visualization and signal analysis in [2]. In [7], the author developed a hyper-complex Fourier transform of a vector field and has applied this to images.

Geometric Algebra (GA) methods were introduced in [8] and [9] for image processing where it was shown that hyper-complex convolution is in fact a subset within GA. Independently, in [10] the GA or Clifford convolution Fourier transform method was applied to pattern matching on vector fields which are used for the visualization of 3-D vector field flows. This work suggested that the convolution technique based on GA is superior because of the unified notation used to represent scalar and vector fields.

A large body of work exists in the general area of color edge detection which is concerned with the relative merits of different basic approaches and quantifying the ability of algorithms to identify features from images [11–15]. For example in [11] it is suggested that leveraging GA is an effective method of detecting edges and that it can also provide a "metric" for the assessment of the strength of those edges. One of the interesting aspects of the proposed work is taking advantage of the simplification of the terms, which has the potential for much simpler implementation, making this an ideal candidate for hardware implementation. We can contrast our approach and also [11] with the more general techniques (Euclidean) described in [12]. In [12] the author evaluates different approaches for feature detection, however in many cases the relevance comes after the basic transformations, such as those presented in our approach or even in [11]. Similarly, [13] discusses the use of visual cues and how the use of an optimally combined set of rules can improve the ability of a vision system to identify edges. However this would be useful in a system environment, again, after the initial transformation had been completed. In a similar manner, [16] describes derived classes of edge detection techniques and quantifies how effective they are in practice, particularly for natural images. The work described in [16] also highlights the possibilities for implementing vision systems that target specific operations or transformations such as GA, where significant performance benefits can accrue, and integrating this with a general

purpose processor that can then leverage other algorithms taking advantage of the results of the efficient processing completed by the partner GA processor. If we consider the general area of research using GA techniques, in most cases the assumption is made that any algorithm can be implemented in general purpose hardware. Most real world scenarios (such as dynamic object recognition) require real time operation, making a relatively slow software solution based on a general purpose platform impractical.

In our proposed approach, the above ideas have been extended further by introducing color vector transformations and a difference subspace within GA. Based on certain properties of these transformations, a hardware architecture to compute all the different GA products has been proposed. The experimental results show the use of the GA methods and proposed hardware for three different edge detection algorithms.

This paper is organized as follows. Section 2 establishes some important GA fundamentals and introduces the techniques for implementing rotations in 3-D. This section also demonstrates how the GA and rotation techniques can be applied to the topics of transformation and difference subspace. This introduction therefore establishes the theoretical foundation of the relationship between GA operations and how the transformations in the GA framework can be exploited to identify chromatic or luminance shifts in an image when the color changes. Section 3 introduces the details of the proposed GA Micro Architecture (GAMA) designed specifically for this application, describes the custom ASIC implementation and reports the experimental results for this hardware implementation, with a focus on the geometric operations per second, and instructions per second, providing a comparison with other hardware and software implementations. Section 4 demonstrates how the technique can be extended to use rotor convolution for color edge detection, with examples including color "block" images and natural images. Section 5 and section 6 extends this to color sensitive edge detection and color smoothing filter, respectively. Finally, concluding remarks are given in Section 7.

2. Geometric Algebra Fundamentals, Rotations and Transformations

2.1. Geometric Algebra (GA) in 3-D

GA has proven to be an extremely powerful and flexible approach for applying complex geometric transformation to objects in both software and hardware applications. One of the major advantages of GA is that once expressions have been defined for relatively low order systems (for example 2-D or 3-D) it becomes straightforward to extend these to much higher order systems. Expressions within GA embed and extend existing theories and methods to express geometric relations without the need for special case considerations in higher dimensions [17–20]. The key to this approach is in how the GA framework handles vectors of different types, and therefore in order to understand some of the key concepts in this paper, it is useful to describe the fundamentals of how vectors are handled in a GA context, particularly with reference to the conventional Euclidian space.

Consider the Euclidean 3-D vector space E^3, which is defined by the orthonormal basis vectors e_1, e_2 and e_3. This 3-D space in E^3 can be decomposed into an eight dimensional real vector space having the eight elements in G^3 shown in Equation (1). The elements of this algebra or real vector space are called the multivectors, and essentially describe all the possible geometric objects within that vector space including scalars (with no vector), vectors (lines), bivectors (surfaces) and trivectors (volumes).

The concept is useful in that any or all of the individual elements can be considered together as a single entity called a multivector.

$$\underbrace{1}_{\text{scalar}} \underbrace{e_1, e_2, e_3}_{\text{vector}} \underbrace{e_1 e_2, e_2 e_3, e_3 e_1}_{\text{bivector}} \underbrace{e_1 e_2 e_3}_{\text{trivector}} \quad (1)$$

Fundamental algebraic rules exist for objects defined within a GA framework. Multiplication of elements within GA is associative, bilinear and commutative for scalar and trivectors but anticommutative for bivectors and is defined by the rules in Equation (2).

$$\begin{gathered} 1e_i = e_i \; i = 1,2,3 \\ e_i e_i = 1 \; i = 1,2,3 \\ e_i e_j = -e_j e_i \; i,j = 1,2,3, i \neq j \end{gathered} \quad (2)$$

As already discussed, within GA it is possible to add or multiply different vector elements to form a multivector. For example, a generic multivector in 3-D is the linear combination of the fundamental eight elements shown in Equation (1) and is defined by Equation (3).

$$\underbrace{1}_{\text{scalar}} + \underbrace{e_1 + e_2 + e_3}_{\text{vector}} + \underbrace{e_1 e_2 + e_2 e_3 + e_3 e_1}_{\text{bivector}} + \underbrace{e_1 e_2 e_3}_{\text{trivector}} \quad (3)$$

Within GA the multiplication of any two multivectors a and b results in the geometric product, which consists of the inner product or dot product ($a \cdot b$) and the outer product ($a \wedge b$) and is given by Equation (4).

$$\text{ab} = a \cdot b + a \wedge b \quad (4)$$

The inner product or the dot product, gives the magnitude of the vectors and the outer product $a \wedge b$ gives the orientation of the plane or the oriented area that is formed by sweeping the vectors a and b. The geometric product is the most important element of this algebra and all the other meaningful operations are derived from it [17]. Clearly there are a large number of possible operations that can be carried out using this framework; however, we are particularly interested in the transformations that become possible within GA, especially rotations. The next section therefore describes how 3-D rotations are performed using GA.

2.2. Rotations in the Geometric Algebra 3-D Space

In Geometric Algebra, a rotor R is an element which is used to rotate any vector within the 3-D space and satisfies the relation: $R\tilde{R} = 1$, where $\tilde{R}$ is the conjugate of R. One of the useful aspects of the GA framework is that if only the bivectors of the algebra are used, it can be shown that the quaternions are a subset of the GA [20]. If $\mathcal{F} = [a_0, a_1, a_2, a_3]$ is defined as a unit quaternion then the one to one mapping between the quaternion and the rotor which performs the same rotation in GA is given by Equation (5) (where I = $e_1 e_2 e_3$ is the pseudoscalar).

$$\text{R} = \underbrace{a_0}_{\text{scalar}} + \underbrace{a_1 \text{I} e_1 - a_2 \text{I} e_2 + a_3 \text{I} e_3}_{\text{bivectors}} \quad (5)$$

Therefore taking only the scalar and bivector parts, a general rotation in 3-D [8], [18] can be written as shown in Equation (6).

$$R = exp(B\frac{\theta}{2}) = \cos\frac{\theta}{2} + B\sin\frac{\theta}{2} \quad (6)$$

Where θ represents a rotation about an axis parallel to unit bivector B and the direction of rotation axis is given by $\mu_1 e_2 e_3 + \mu_2 e_3 e_1 + \mu_3 e_1 e_2$ spanned by the bivector basis. We will show in this paper how this rotational element R is an important element while discussing the transformation subspace for color images in the following section. Also, a detailed discussion following an example is discussed in Section 4.3.

2.3. Transformation and Difference Subspace

In the context of this work, it is assumed that the perceived color is a vector in the 3-D Euclidean space and not as separate *r*-*g*-*b* image planes. In this regard the bivector representation of color vectors in GA fits neatly for the 3-D Euclidean space. Using this approach the (*r*,*g*,*b*)vector of the color image $c_{m,n}$ can be written as shown in Equation (7).

$$c_{m,n} = r_{m,n}e_2e_3 + g_{m,n}e_3e_1 + b_{m,n}e_1e_2 \quad (7)$$

where $r_{m,n}$, $g_{m,n}$ and $b_{m,n}$ are the *rgb* vectors of the image $c_{m,n}$, where m and n are the row and column pixels, respectively. In this section the color is defined as a vector or a single entity (Equation (7)) and the image is treated as a superset of this entity. In the later experimental sections we will show by doing this how the image processing algorithms become straightforward and intuitive.

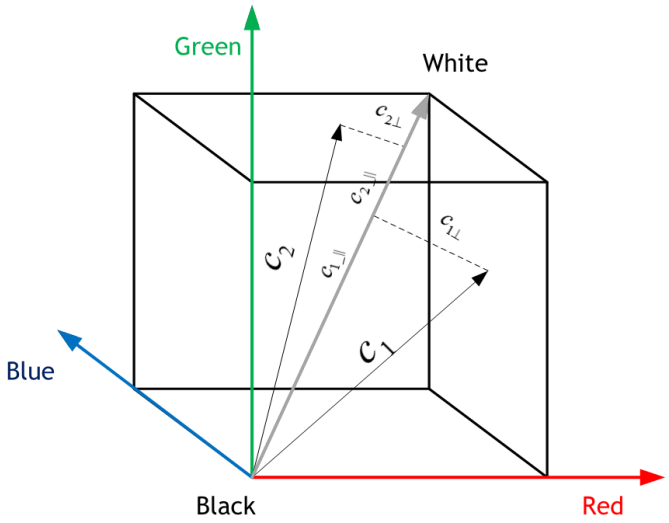

Figure 1. RGB vectors and the color cube.

For this discussion, μ is the diagonal axis, also the gray line in the color cube (Figure 1) is represented as in Equation (8).

$$\mu = e_1 e_2 + e_2 e_3 + e_3 e_1 \quad (8)$$

For $r = g = b$ the pixel is achromatic in nature and is represented as a gray line. For transformation a normalized color representation is chosen for clarity. This ensures that the orientation information is

kept while the distance information is normalized. For a unit transformation on the normalized color, the rotation vector is expressed as shown in Equations (9) and (10).

$$\mathrm{R} = \cos\theta + \frac{1}{\sqrt{3}}\mu\sin\theta = \cos\theta + \frac{e_{12} + e_{23} + e_{31}}{\sqrt{3}}\sin\theta \tag{9}$$

$$\widetilde{\mathrm{R}} = \cos\theta - \frac{1}{\sqrt{3}}\mu\sin\theta = \cos\theta - \frac{e_{12} + e_{23} + e_{31}}{\sqrt{3}}\sin\theta \tag{10}$$

The rotation given by R and $\widetilde{\mathrm{R}}$ rotates any vector by an angle $\frac{\theta}{2}$ in 3-D about an axis parallel to the rotation axis. The unit transformation on a color element C (Equation (7)) is given by Equation (11), where $C = re23 + ge31 + be12$.

$$\begin{aligned} R\mathrm{C}\widetilde{\mathrm{R}} &= (cos\theta + \frac{1}{\sqrt{3}}\mu sin\theta)(\mathrm{C})(cos\theta - \frac{1}{\sqrt{3}}\mu sin\theta) \\ &= \underbrace{cos^2\theta(\mathrm{C})}_{\mathrm{I}} - \underbrace{\frac{cos\theta sin\theta}{\sqrt{3}}(\mathrm{C})\mu}_{\mathrm{II}} + \underbrace{\frac{cos\theta sin\theta}{\sqrt{3}}\mu(\mathrm{C})}_{\mathrm{III}} + \underbrace{\frac{sin^2\theta}{\sqrt{3}}\mu(\mathrm{C})\mu}_{\mathrm{IV}} \end{aligned} \tag{11}$$

Part II in Equation (11) reduces to

$$-(r + g + b) - [(b - g)e_{23} + (r - b)e_{31} + (g - r)e_{12}] \tag{12}$$

and III reduces to

$$-(r + g + b) + [(b - g)e_{23} + (r - b)e_{31} + (g - r)e_{12}] \tag{13}$$

and part IV reduces to

$$(e_{23} + e_{31} + e_{12})(re_{23} + ge_{31} + be_{12})(e_{23} + e_{31} + e_{12}) \tag{14}$$

this becomes:

$$re_{23} + ge_{31} + be_{12} - 2(re_{31} + re_{12} + ge_{23} + ge_{12} + be_{23} + be_{31}) \tag{15}$$

Using the reductions in Equations (12) to (15), Equation (11) can therefore be rewritten as shown in Equation (16):

$$\begin{aligned} &\underbrace{\cos2\theta(re_{23} + ge_{31} + be_{12})}_{\mathrm{A}} + \underbrace{\frac{2}{3}\sin^2\theta\mu(r + g + b)}_{\mathrm{B}} \\ &\quad + \underbrace{\frac{1}{\sqrt{3}}\sin2\theta[(b - g)e_{23} + (r - b)e_{31} + (g - r)e_{12}]}_{\mathrm{C}} \end{aligned} \tag{16}$$

The term "A" in the above equation is the *rgb* space component, "B" is the intensity component and the "C" term is the color difference or the chromaticity (hue and saturation) of the vector.

If the vector is rotated by an angle $\theta = \frac{\pi}{4}$, the above equation reduces to only two components as shown in Equations (17) and (18), where the space component "A" is cancelled. This transformation is RGB to HSI conversion where the two components in the equation describe the luminance and chrominance of the image. Therefore, for $\theta = \frac{\pi}{4}$.

$$R\mathrm{C}\tilde{R} = \underbrace{\frac{1}{3}\mu(r + g + b)}_{\text{luminance}} + \underbrace{\frac{1}{\sqrt{3}}[(b - g)e_{23} + (r - b)e_{31} + (g - r)e_{12}]}_{\text{chrominance}} \tag{17}$$

where r, g, b are scalar quantities. Similarly the opposite rotation is given by Equation (18).

$$\tilde{R}CR = \underbrace{\frac{1}{3}\mu(r+g+b)}_{\text{luminance}} - \underbrace{\frac{1}{\sqrt{3}}[(b-g)e_{23} + (r-b)e_{31} + (g-r)e_{12}]}_{\text{chrominance}} \tag{18}$$

If the color vector is homogeneous then addition of two transforms equate to:

$$RC\tilde{R} + \tilde{R}CR = \frac{2}{3}\mu(r+g+b) \tag{19}$$

this results in the intensity of the image. Subtracting the two transforms gives:

$$RC\tilde{R} - \tilde{R}CR = \frac{2}{\sqrt{3}}[(b-g)e_{23} + (r-b)e_{31} + (g-r)e_{12}] \tag{20}$$

which is the difference of the color vector, and this is often referred to as the change in chromaticity or a shift in hue. It can therefore be concluded that if the color vectors are different then the above transformations do not cancel out and this will result in a hue or chromatic shift. On the other hand a homogeneous color vector results in an intensity component that generates sharp edges when a color change occurs within an image. This is an extremely powerful and interesting result, which can be applied to edge detection in hardware if the required GA transformations are available.

3. Rotor Edge Detection Using Geometric Algebra Co-Processor

The previous section showed that in order to implement color edge detection using rotor transformations the main GA computations required are the geometric product, multi-vector addition and subtraction. From an implementation perspective, these functions can also be decomposed into parallelized tasks, each task involving Multiply and Accumulation (MAC) operations, which can map directly to specific hardware. It is already known that significant speed advantages can be gained when GA is implemented in hardware instead of software [21–23]. The hardware architecture is discussed in this section, however, a detailed description is beyond the scope of this paper, and the reader is referred to [23] for more information. An important advantage of the architecture is that by having dedicated hardware functions to calculate the GA operations directly, significant efficiencies can be achieved over general-purpose hardware. For example, although a general-purpose processor may have dedicated GA software, the underlying transformations will still use standard processing resources, which will be less efficient than a hardware, which directly maps the GA software operations.

3.1. Hardware Architecture Overview

The proposed rotor edge detection hardware architecture consists of an IO interface, control unit, memory unit and a central Geometric Algebra Core [21–23] consisting of adder, multiplier, blade logic and a result register (Figure 2). The architecture supports both single and double precision floating-point numbers, four rounding modes, and exceptions specified by the IEEE 754 standard [24]. This can be seen as effectively a GA Co-Processor, which operates in conjunction with a conventional general-purpose microprocessor.

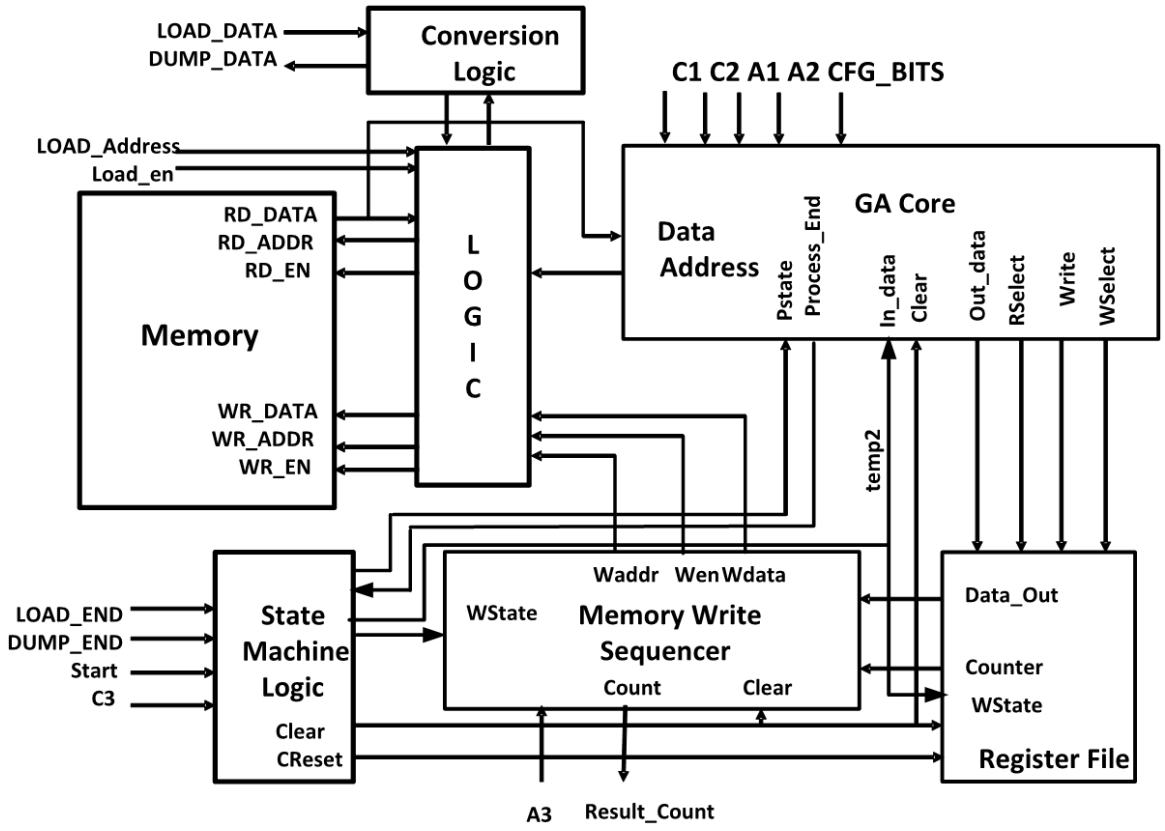

Figure 2. Rotor edge detection geometric algebra co-processor architecture with permission from the authors of ref [23].

The floating-point multiplier is a five-stage pipeline that produces a result on every clock cycle. The floating point adder is more complex than the multiplier and involves steps including checking for non-zero operands, subtraction of exponents, aligning the mantissas, adjusting the exponent, normalizing the result, and rounding off the result. The adder is a six-stage pipeline that produces a result on every clock cycle. It is important to state here that the proposed hardware can process other products of Geometric Algebra with ease.

The state machine governing the processing stages of Geometric Algebra has six states, idle, clear, load, process, write, and memory dump. Firstly, the idle state waits for the start signal to be high to trigger the state machine. After that, the state machine will come into the clear state where it clears any registers and then to the load state to export load as "1" to load input data into the registers. Then it processes the result in the required clock cycles based on the control word (cfg_bits). Finally, it outputs the product in the output state. Apart from the transformation of the load state, which is triggered by the start signal, the others just proceed to next state automatically after an expected number of cycles based on the control word.

The long 320-bit word datapath is coordinated by controller (LOGIC) and sequencer unit. The transfer of the data is done in the input and output interface unit (conversion logic). The signals are all defined as input and output registers to the system architecture. Selection of the data input and output is based on the 16-bit control word. The control bits are used for configuring (cfg_bits) the processing core for different operations. These control bits are responsible for defining the data interface and configuring

the operators at the correct time and outputting the result. The control block along with the sequencer ensures effective queuing and stalling to balance the inputs in different stages in the datapath.

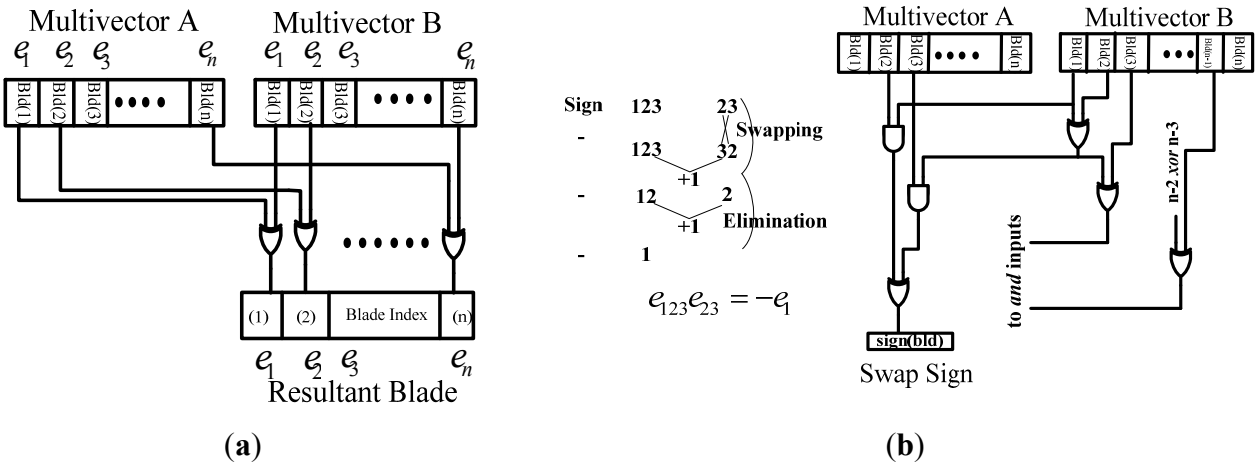

Figure 3. Computing and swapping of basis blades. (**a**) Basis vector computation; (**b**) Swap computation.

3.2. Blade Logic

Another important element of this architecture is the blade computation. The "blade" is simply the general definition of vectors, scalar, bivectors and trivectors. The blade index relationship is defined by the elements being computed, in a similar way that matrix operations result in an output matrix dimension that is defined by the input matrix dimensions. For example, to compute e_1 with e_2, the resultant blade index is e_1e_2. Similarly if we multiply e_1e_2 with e_2 then the resultant basis blade index is e_1. This can be implemented by a multiplication table, an approach followed by many software implementations. However, accessing a memory in hardware is a slower operation than a simple EXOR function (Figure 3a). Determining the sign due to the blade index is not straightforward due to the invertible nature of the geometric operation. For example the blade index multiplication of e_1 with e_2 gives e_1e_2 whereas with e_2 and e_1 results in $-e_1e_2$. The resulting circuit which is a cascade of EXOR gates takes care of the swapping of the blade vector and the AND gates compute the number of swaps that the blade element undergoes (see Figure 3b).

Detailed design description of the hardware implementation can be found in [23].

Table 1. GA-Coprocessor/Rotor hardware (area, speed).

Design	No of cells	Area (μm^2)	No of gates	Frequency
Rotor Hardware	35,355	813,504	133,361	130.0 MHz

3.3. ASIC Implementation

The proposed hardware is described using synthesizable VHDL which means that the architecture can be targeted at any technology, FPGA or ASIC. Standard EDA tools perform all the translation from VHDL to silicon including synthesis and automatic place and route. The architecture was synthesized to a CMOS 130 nm standard cell process. The synthesis was carried out using the Synplify ASIC and the

Cadence Silicon Ensemble place and route tools. The synthesis timing and area reports are summarized in Table 1. The chip area was 1.1 × 1.1 mm^2 (inset in Figure 4) and the estimated maximum clock frequency of the design was 130 MHz.

The prototype ASIC was packaged in a standard QFP package and this was then placed on a test printed circuit board (PCB), with power supply connections and a link to a FPGA development kit that supported a general purpose processor for programming and data handling (Figure 4). The test board was then used to evaluate the performance of the GA processor in handling a variety of test data, described in detail in the next section.

3.4. Comparison with Other GA Implementation Hardware and Software

This section describes the experimental verification results. When calculating the GA operations the GOPS (Equation (20)) is particularly important because the designer is then able to determine whether the timing constraint put by the clock cycles and GOPS provided by that particular implementation is relevant.

$$GOPS = \frac{1}{no\ of\ cycles \times clk\ period} = \frac{1}{latency} \tag{21}$$

Table 2 gives a comparison for different dimensions, GA processor performance in MHz, clock cycles, latency and Geometric Operations per Second (GOPS). This also provides a comparison to the GA hardware implementations in [21,23,25–30]. It was found that the proposed hardware is an order of magnitude faster as shown in Table 2 (columns 5–6).

In [27], the normalized GOPS (in Table 2) is found out by dividing the number of MAC units, in this case it was 74. However, the authors have used the hardware resources available in the FPGA to their advantage and shown a threefold performance improvement as compared to the proposed hardware architecture. The GA products are computed on every clock cycle after a specified latency. The performance of their hardware is further improved by the authors by roughly two to three orders of magnitude in a very recent paper [28] as compared to the proposed ASIC architecture discussed here.

Table 2. GA performance figures and comparison with previously published results.

Reference	Frequency (MHz)	HW resources	No of processing cycles	GOPS (in thousands)	GOPS (In thousands, Normalized ‡)
Perwass *et al.* [25]	20.0	1M, 1A (2)	176 [1], 544 [2], 2044 [3]	112.52 [1], 36.75 [2], 9.79 [3]	56.26, 18.37, 4.89
Mishra-Wilson [21]	68.0	2M, 3A (5)	84 [1], 224 [2], 704 [3]	809.6 [1], 303.7 [2], 96.62 [3]	161.92, 60.74, 19.32
Gentile *et al.* [26]	20.0	24M, 16A (40)	56 [2]	357.15	8.9
Lange *et al.* [27]	170.0	74 †	366 [3]	464.5	6.28
Franchini *et al.* [29]	50	24M	56 [1]	892.8	37.2
Franchini *et al.* [30]	100	64M	405 [2], 1180 [3]	246.9 [2], 84.7 [3]	3.8 [2], 1.3 [3]
Mishra *et al.* [23]	65.0	1M, 1A (2)	177 [1], 455 [2], 1399 [3]	367.2 [1], 142.8 [2], 46.4 [3]	183.6 [1], 71.4 [2], 23.2 [3]
This Work	125.0	2M, 3A (5)	84 [1], 224 [2], 704 [3]	1488 [1], 558 [2], 177.5 [3]	297.6 [1], 111.6 [2], 35.5 [3]

[1] GA in 3D; [2] GA in 4D; [3] GA in 5D, M = Multiplier, A = Adder; † Number of DSP48Es in Xilinx (These are Multiply and Add Units); ‡ Normalized to Hardware resources used (GOPS/Hardware).

3.5. FPGA and ASIC Test Results

Subsequently, the ASIC test was performed using a 125 MHz clock (a little less than the maximum clock frequency of 130 MHz). A set of 1 k, 4 k, and 40 k product evaluations was carried out to yield the raw performance (where all multivectors are present) of the hardware. The results for geometric product evaluation at different clock speeds along with the performance measures obtained when the design was targeted to an FPGA family (Xilinx XUPV2P, San Jose, CA, USA) are given in Tables 3 and 4.

The implementation results of the processor core at different dimensions suggest that the performance is comparable to some of the software packages implemented in software (GAIGEN, Amsterdam, The Netherlands) which runs on high speed CPUs. To see how much advantage was gained by this hardware we compared the performance of GABLE [31], GAIGEN [32] and the proposed hardware (see Table 3). To ensure consistency with GABLE, only the three-dimensional GA implementation is considered. For GAIGEN the "e3ga.h" module is used for the performance evaluation.

Table 3. Comparison of different Software and the proposed hardware.

	Software		**FPGA**		**ASIC**
No.	GABLE	GAIGEN	@12.5 MHz	@68 MHz	@125 MHz
1000	0.8910	9.91×10^{-4}	6.92×10^{-3}	1.27×10^{-3}	6.97×10^{-4}
4000	3.5750	3.956×10^{-3}	2.76×10^{-2}	5.08×10^{-3}	2.78×10^{-3}
40,000	35.210	3.978×10^{-2}	0.276	5.08×10^{-2}	2.78×10^{-2}

Tables 3 and 4 show the performance comparison on the software that ran on 2GHz Intel Pentium processors and the proposed hardware. The software implementation GABLE (in MATLAB) was found to be extremely slow however the GAIGEN software running on a CPU was found to be of comparable performance to our proposed hardware (Table 3). However, when the number of elements is varied, e.g., (vectors multiplied with a scalar and bivector) $((e_1 + e_2 + e_3) \times (1 + e_{12} + e_{23} + e_{31}))$ it resulted in a performance increase of more than 3× when compared to the software implementation as shown in Table 4.

Table 4. Comparison of Software and the hardware for vector and bivector calculations.

	Software		**FPGA**		**ASIC**
No.	GABLE	GAIGEN	@12.5MHz	@68MHz	@125MHz
1000	0.8920	0.001013	0.00308	0.000567	0.000312
40000	35.1910	0.040619	0.12320	0.022638	0.012480

Furthermore, it was seen from the experiments that the number of cycles can be optimized by hardwiring the operators which are constant over the operation. For example, for a windowing operator typical to an image-processing algorithm, four of the *multivectors* which define the window or the filter can be hardwired. This leads to a 25% savings in the processing cycle due to the savings associated with I/O transfers every operation. The overall performance gains using the GA Co-Processor over existing software [31,32] approaches are more than 3.2× faster than GAIGEN [31] and more than 2800× faster than GABLE [32]. The above comparison shows that the proposed hardware provides speedup and

performance improvement over the existing hardware (Table 2) as well the software implementations on CPU.

In Section 2, the basic technique of rotor detection has already been discussed. This technique is applied to color difference edge detection on our proposed hardware in the next section.

The proposed hardware architecture supports the general computations involving GA operations. To show the usefulness of the framework, we chose an image processing application in general and edge detection algorithm in particular. In this application, we demonstrate that even if the data is not a full multivector containing all the elements, one could still take advantage of the architecture with very little changes to the way the data is fed into the core. We also demonstrate the performance with two cores. The platform can be used for signal processing, vision and robotics applications.

4. Color Difference Edge Detection Using a Geometric Algebra Co-Processor

4.1. Introduction

A conventional edge detection process involves convolving masks $m_L(x,y)$ (left) and $m_R(x,y)$ (right) of the size $X \times Y$ with the image $c(m,n)$ of dimension $(m \times n)$ (Equation (22)). In a GA based approach, convolution masks undergoing the following equation is similar with the exception that the scalar quantities are replaced with multivector masks (Equations (23) and (24)).

$$\hat{c}(m,n) = \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} m_L[x,y]\, c[m-x, n-y]\, m_R[x,y] \tag{22}$$

Previous work [33] has already reported the usefulness of hypercomplex masks for edge detection and this was extended to rotor convolution by Corrochano-Flores in [8]. The rotor convolution works exactly the same way as the hypercomplex convolution and operates on the color vectors of the image. The horizontal left and right masks for the rotor convolution are defined in Equations (23) and (24). The vertical masks are obtained by interchanging the rows and columns of the two masks.

$$m_L = \begin{bmatrix} R & R & R \\ 0 & 0 & 0 \\ \tilde{R} & \tilde{R} & \tilde{R} \end{bmatrix} \tag{23}$$

$$m_L = \begin{bmatrix} \tilde{R} & \tilde{R} & \tilde{R} \\ 0 & 0 & 0 \\ R & R & R \end{bmatrix} \tag{24}$$

where the rotors are given by Equation (25).

$$R = se^{n\pi/4} = s[\cos(\pi/4) + \mu \sin(\pi/4)] \tag{25}$$

In the above equation, μ is the unit vector and is given by $\mu = (e_2 e_3 + e_3 e_1 + e_1 e_2)/\sqrt{3}$ and $s = 1/\sqrt{6}$ is the scale factor. The left and right masks are applied to each of the color pixels, which give the following convolution in the simplified form [9]. Therefore convolution in Equation (22) results in Equation (26).

$$\begin{aligned} &\hat{c}(m,n) = R[\mathbf{C}(m,n)]\tilde{R} \\ &= R(c_{m-1,n-1} + c_{m-1,n} + c_{m-1,n+1})\tilde{R} \; + \; \tilde{R}(c_{m+1,n-1} + c_{m+1,n} + c_{m+1,n+1})R \end{aligned} \tag{26}$$

Here the color is taken as a single vector and is split in parallel and perpendicular to the rotation axis. In the 3D color cube (Figure 1, the rotation axis is the unit vector. As shown in Equation (26), if the colors are uniform in upper and lower rows, then the amount of the rotation to the perpendicular component for both the rows will be same. Therefore they cancel out and the resultant vector lie on the gray axis. That will indicate no edges present in the image. In case of dissimilar regions, *i.e.*, the upper and lower rows having different values, the perpendicular component rotated will be different and when added they will not cancel each other. The resultant vector will lie somewhere else in the cube, off the gray axis—indicating the presence of an edge.

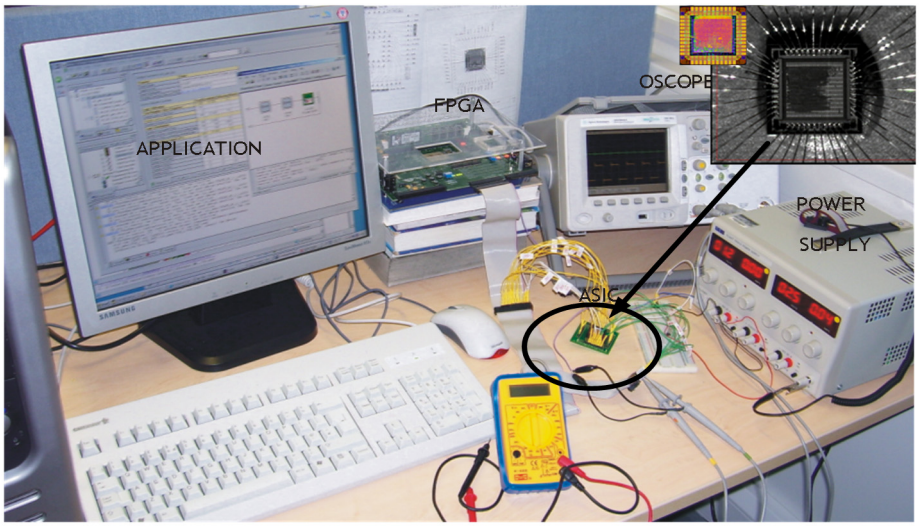

Figure 4. Actual test setup (inset Fabricated ASIC Micrograph).

4.2. Experimental Configuration

The experimental setup for the convolution operation involving hardware is as shown in Figure 4. It consists of the two blocks (to bin matlab, to bmp matlab block in the Figure 5) and the ASIC (hardware) mounted on a PCB. The images were first read in MATLAB (Mathworks, Natick, MA, USA and converted into binary images (processed off line) and were then fed to the hardware. Then the binary results from the hardware were again converted to the bitmap (bmp) images and are processed off line. Three images (Figures 6a, 7a and 8a) of different image sizes (128 × 128, 256 × 256 and 512 × 512) were taken. The masks defined in [8,33] were applied to all three images.

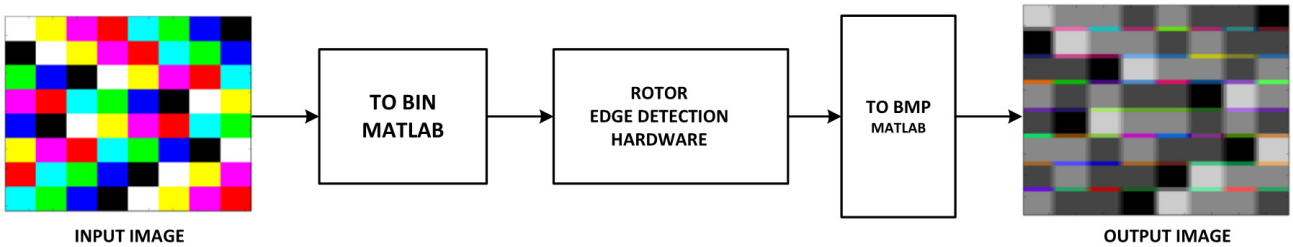

Figure 5. Offline processing with the GA Coprocessor hardware.

4.3. Image Convolution Results

The hardware performs all the necessary geometric calculations and generated the results. The results of the convolution of the three images are shown in Figures 6b, 7b and 8b.

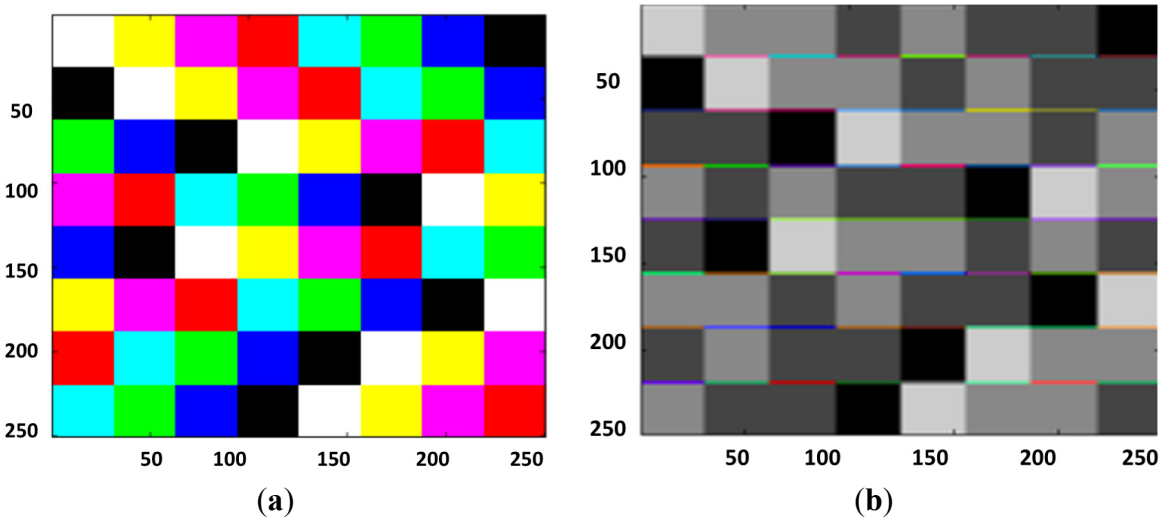

Figure 6. (**a**) Original Color Block and (**b**) Color Block after Rotor Convolution. With permission from: S. J. Sangwine) http://privatewww.essex.ac.uk/sjs/research; (with permission from J. Kominek) http://links.uwaterloo.ca/bragzone.base.html.

Rotor convolution using Equation (26) was applied on the test image of the "color block" which has an 8 × 8 array of colored squares (see Figure 6a). The result of the filtered image is shown in Figure 6b. The filtered image has gray areas where the squares had uniform color. But it has colored lines at the edges or where there was a change of color. It was also observed at the edges between the black and white blocks.

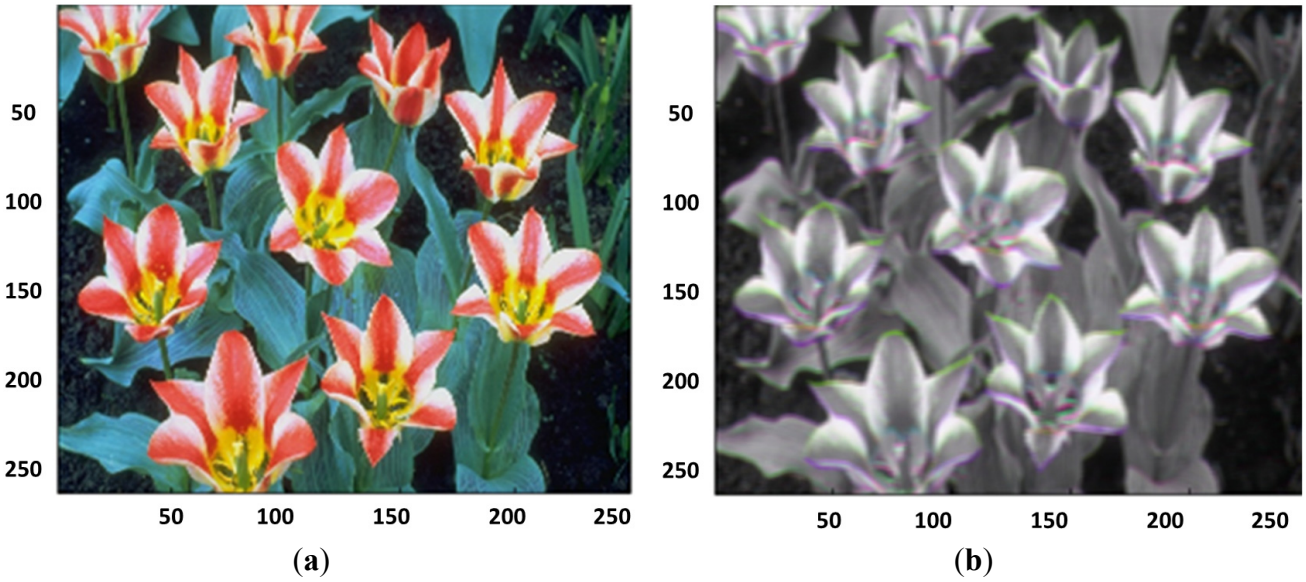

Figure 7. (**a**) Original tulip image and (**b**) Tulip image after rotor convolution.

In this approach the color is considered to be a vector and expressed as a single entity. As shown in Figure 1, the color vector (**C**) is split into two components; $\mathbf{C}_{\parallel}$, which is the parallel component to the

gray axis, and $\mathbf{C}_{\perp}$ is the perpendicular component. When the masks are applied on the color vector only the perpendicular component $c_{1\perp}$ is affected but the parallel component $c_{1\parallel}$ is unchanged. The convolution rotates the perpendicular component by an amount specified by the rotor angle θ. Since rotor R rotates any vector in a clockwise manner, the rotor $R\mathbf{C}\tilde{R}$ would rotate the color vector by the same amount as would the rotor $\tilde{R}\mathbf{C}R$ but with an opposite direction. Hence if the color vectors are homogeneous then both the components would cancel. This point or pixel would fall on the gray axis and the pixel would be perceived as a gray picture or the intensity of the image.

However, if the color components are *not* homogeneous then the color vector will be rotated by an unequal amount by the two rotors. Thus the resultant vector would lie somewhere else in the color cube, far from the gray axis (Figure 1). For $\theta = \pi/2$, the $\tilde{R}\mathbf{C}R$ rotates the color by $\pi/2$ and $R\mathbf{C}\tilde{R}$ by $-\pi/2$. Hence the two color vectors cancel each other due to the rotation operation when they are uniform and fall on the black and white axis. Otherwise they fall outside this gray line where $r = g = b$, giving a color value to the pixel. This signifies that the rotor operation is a shift in hue (chromaticity) of the image. In the areas where the upper and lower pixels are similar the rotors produce a gray scale image. When these pixels differ in color, the rotors produce different colors, as they do not cancel in the chromatic sense. Hence the change of direction due to rotation of colors results in different colors on the edges. This type of change is also be observed on the filtered images of tulips (Figure 7b) and top edges of the hat of the "Lena" image (Figure 8b), and the edges of flowers of the "Tulip" image.

Figure 8. (**a**) Original lena image and (**b**) Lena image after rotor convolution.

4.4. Discussion of Rotor Convolution Results

From the experiments using the proposed hardware, it was observed that for an image size of 128 × 128 pixels, the total number of geometric product multiplications is 1.96×10^5. Each color pixel is treated as a vector and each convolution operation consists of 12 geometric product multiplications (GP mul) and four geometric product additions (GP add). For image size of 256 × 256 and 512 × 512, the number of multiplication and addition increases proportionally.

It was observed that further optimization of these algorithms is also possible. For example, four additions followed by four convolutions on these vectors can be used with the same result due to

the linearity property of the rotor operations. This results in huge savings in terms of geometric operations where the hardware is designed to handle such specialized computations efficiently. As shown in the Table 5, the total time taken for the convolution of a 128 × 128 image takes 5.70×10^6 cycles which amounts to 45.6 ms (third column, third row of Table 5) based on the hardware operating frequency set at 130 MHz. The convolution times for different sized images are given in the following Table 5.

Table 5. Four GP for rotor convolution.

Image Size	Full Multivector		ASIC with single core		ASIC with two Core	
Image	Cycles	T (ms)	Cycles	T (ms)	Cycles	T (ms)
128 × 128	5.70×10^6	45.6	4.98×10^6	38.3	2.55×10^6	19.6
256 × 256	2.39×10^7	184	1.99×10^7	153.2	1.02×10^7	78.6
512 × 512	9.19×10^7	735	7.96×10^7	613.0	4.08×10^7	314.5

The hardware consists of two multipliers and three adders (labeled as ASIC with two cores in the Tables 5 and 6) in an optimized structure already discussed in the previous section. Tables 5 and 6 show the time required for each calculation running on this hardware. Entries in the Table 5 (columns 2 and 3) show the results for a 3-D vector space consisting of an eight multivector, *i.e.*, a full multivector (MV). Therefore the multiplication of two multivectors would result in 64 products and 56 additions. However, as discussed earlier, the color is expressed with three bivector elements, which reduces significantly the number of processor cycles while computing the geometric product and geometric additions. The reduction is evident from the number of clock cycles and time column in the Table 5 (columns 2–3 compared to 4–5). The entries in columns 6 and 7 show the performance of the geometric product and additions with two cores (labeled as ASIC with two Core in the table), consisting of three adders and two multipliers.

In Tables 5 and 6, the timing of 4 GP mul and 5 GP add (one extra final addition done, therefore 5 GP Add instead of 4 GP Add) is given, respectively. For the convolution of the image the worst-case timing between the two is considered. For example for an image size 128 × 128, the GP mul is 19.6 ms and GP add is 21.4 ms. In this case only the worst of the two, *i.e.*, 21.4 ms, is taken into account when showing the convolution operation with the slack being wasted as a stall operation in the hardware.

Table 6. Five geometric additions for rotor convolution.

Image Size	Full Multivector		ASIC with single core		ASIC with two Core	
Image	Cycles	T (ms)	Cycles	T (ms)	Cycles	T (ms)
128 × 128	5.89×10^6	45.3	3.27×10^6	25.2	2.78×10^6	21.4
256 × 256	2.35×10^7	181.4	1.31×10^7	100.8	1.11×10^7	85.7
512 × 512	9.43×10^7	725.9	5.24×10^7	403.2	4.45×10^7	342.8

From the experiments we observed that if the color image was treated as a full multivector, a lot of clock cycles were wasted as either stall operations or no operations. If the color vector is expressed as a bivector containing four vectors, there is an immediate performance gain both with single core and two cores. For example in Tables 5 and 6, we can see a 16% gain in speed in the single core. This also occurs when color is expressed as full multivector (45.6 ms) resulting in improvements using a single core (38.3 ms) and finally a 53% gain with two cores (21.4 ms). A similar positive trend is observed for larger image sizes as shown in Tables 5 and 6.

Now that we have demonstrated the approach for color edge detection, we can extend this to the special case of color sensitive edge detection in the following.

5. Color Sensitive Edge Detection—Red to Blue—Using the Geometric Algebra Co-Processor

In this section, the detection of homogeneous regions of particular colors C_1 to C_2 ($C_1 \rightarrow C_2$) are discussed. The edges between these two colors are determined by the GA methods and are an extension of the method discussed in [34]. To show the feasibility of this approach we have considered only synthetic images with strict thresholding criterion to find the edges are considered because the thresholding criterion for synthetic images is straightforward.

Let $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ be two color bivectors. μC_1 is a normalized color bivector of C_1, where μ is given by Equation (27).

$$\mu = \frac{e_2 e_3 + e_3 e_1 + e_1 e_2}{\sqrt{3}}. \tag{27}$$

To find the discontinuity between two regions C_1 to C_2 the convolution can be used with the following hypercomplex filter.

$$m_L = \frac{1}{\sqrt{6}}\begin{bmatrix} \mu C_1 & \mu C_1 & \mu C_1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{28}$$

$$m_R = \frac{1}{\sqrt{6}}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ \mu C_2 & \mu C_2 & \mu C_2 \end{bmatrix} \tag{29}$$

Therefore the convolution operation on an image $c(m \times n)$ of size $m \times n$ is given by:

$$\begin{aligned} \hat{c}(m,n) &= m_L[c(m,n)]m_R \\ &= \sum_x \sum_y m_L(x,y)[c(m-x,n-y)]m_R(x,y) \end{aligned} \tag{30}$$

where x, y are horizontal and vertical masks of size 3 × 3 as given in Equations (28) and (29), respectively (see Figure 9). The above convolution operation results in non-zero scalar part and zero vector part in special cases. For example, to observe the discontinuity from $C_1 \rightarrow C_2$ (red to blue), let C_1 be $e_2 e_3$(red) and C_2 be $e_3 e_1$(blue):

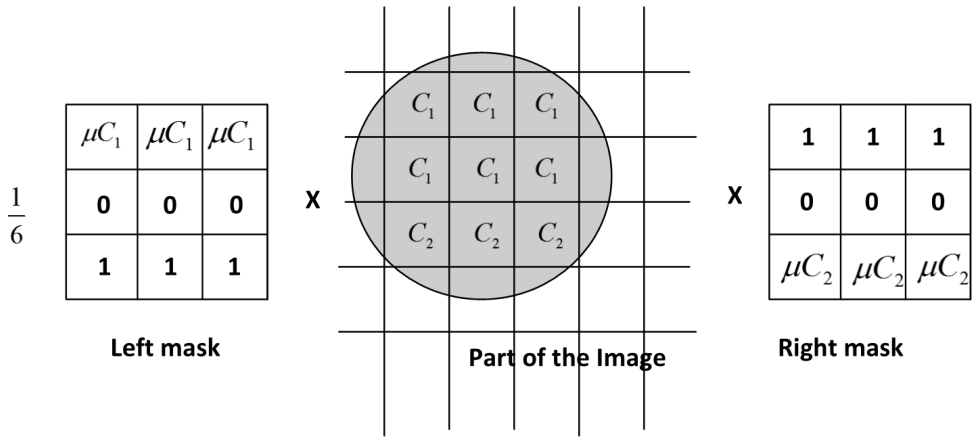

Figure 9. Red to Blue, Convolution by left and right mask.

Normalizing both the vectors, $\mu C_1 = e_2 e_3$ and $\mu C_2 = e_3 e_1$, the convolution results in a scalar value $\frac{1}{6}(-3-3) = -1$ or generally expressed as:

$$\frac{1}{2}(|C_1| + |C_2|) \tag{31}$$

It can be observed that the above convolution results in a non-zero scalar value and zero vector value. As shown in Figure 10a there is only one block where the red to blue ($C_1 \rightarrow C_2$) color transition occurs. When convolving with the mask (Equation (30)) the vector part will be zero. In the above example, the non-zero scalar value is the intensity of the pixel shown in the whiter regions in Figure 10b. In all other places of the image the scalar and vector will have non-zero value. Also, it should be noted that the masks are directional in nature. Therefore, we observe the white line (in the region of interest) when the red to blue occurs and not when the blue to red change occurs.

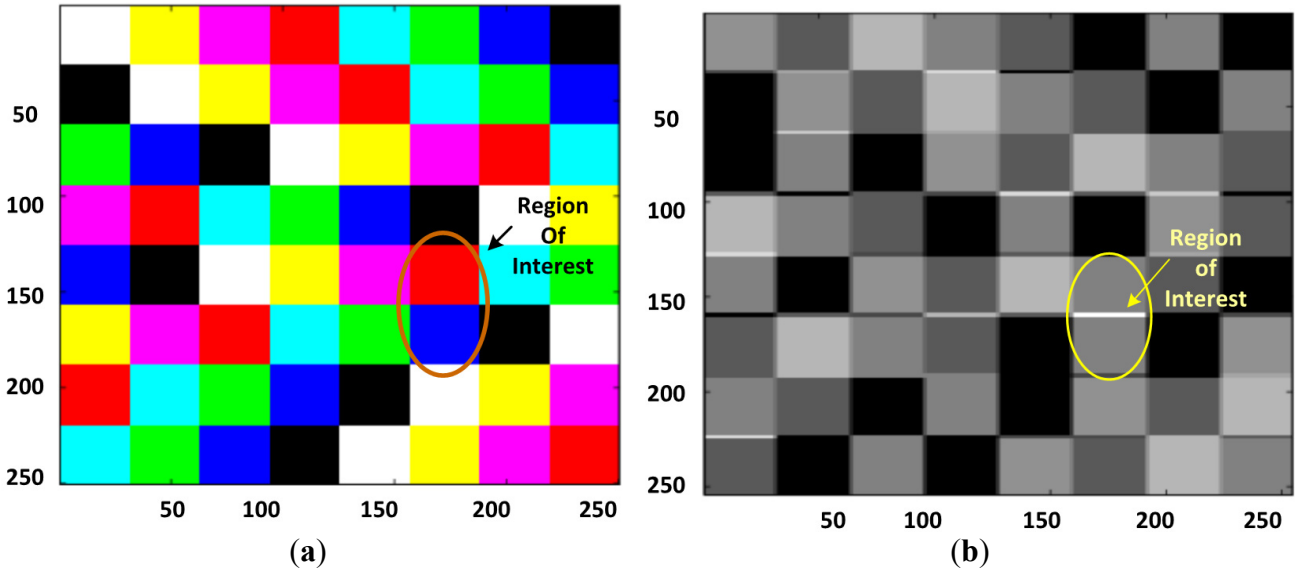

Figure 10. (**a**) Block Image and (**b**) After Step filtering.

Testing of Rotor Convolution in Hardware

Table 7 shows the timing and the clock cycles the hardware takes for the GA geometric product and Table 8 for the GA addition operations. The results also show varying timing for full multivector and timing with single and two cores for the color bivector. Again in this case only the worst of the two (Geo Product and Geo Addition) is taken into account when showing the convolution operation (marked gray in Table 8, using only two cores).

Table 7. Two geometric products for rotor convolution.

Image Size	Full Multivector		ASIC with single core		ASIC with two Core	
Image	Cycles	T (ms)	Cycles	T (ms)	Cycles	T (ms)
128 × 128	2.85×10^6	22.8	2.49×10^6	19.1	1.27×10^6	9.8
256 × 256	1.19×10^7	92.0	9.96×10^6	76.6	5.11×10^6	39.3
512 × 512	4.59×10^7	367.5	3.98×10^7	306.5	2.04×10^7	157.2

Table 8. Four geometric additions for rotor convolution.

Image Size	Full Multivector		ASIC with single core		ASIC with two Core	
Image	Cycles	T (ms)	Cycles	T (ms)	Cycles	T (ms)
128 × 128	4.71×10^6	36.2	2.62×10^6	20.1	2.22×10^6	17.1
256 × 256	1.88×10^7	145.1	1.04×10^7	80.6	8.91×10^6	68.5
512 × 512	7.54×10^7	580.7	4.19×10^7	322.6	3.56×10^7	274.2

6. Color Sensitive Smoothing Filter

Another application of GA in image processing is to apply a low pass filter technique. The smoothing filter can be applied to smooth the color image component in the direction of any color component C_1.

Extending low pass filtering applied to gray scale techniques provides limited color sensitivity as only one color band is convolved with the filter [2]. If all the color bands are considered, then with traditional filtering all the bands will be smoothed equally which is undesirable. With GA, such affine transformation with low pass filtering is achieved in one step [35].

The smoothing of red or cyan components is performed by the following mask:

$$mask_{lpf} = \frac{1}{9}\begin{bmatrix} \mu C_1 & \mu C_1 & \mu C_1 \\ \mu C_1 & \mu C_1 & \mu C_1 \\ \mu C_1 & \mu C_1 & \mu C_1 \end{bmatrix} \tag{32}$$

where μC_1 is the normalized color vector.

The convolution is used to find the homogeneous regions within the image will result in a non-zero scalar part and a zero vector part for the homogeneous regions where the match is found. In all the other places the masking operation results in a non-zero scalar and vector part. Convolution due to this mask results in smoothing along the C_1 direction. The smoothing of red or cyan components, which are parallel to C_1, is changed but the other two perpendicular components remain untouched. Furthermore, the left convolution is different to the right convolution. Hence addition due to left and right convolution will result in cancellation of the vector part leaving only the scalar part.

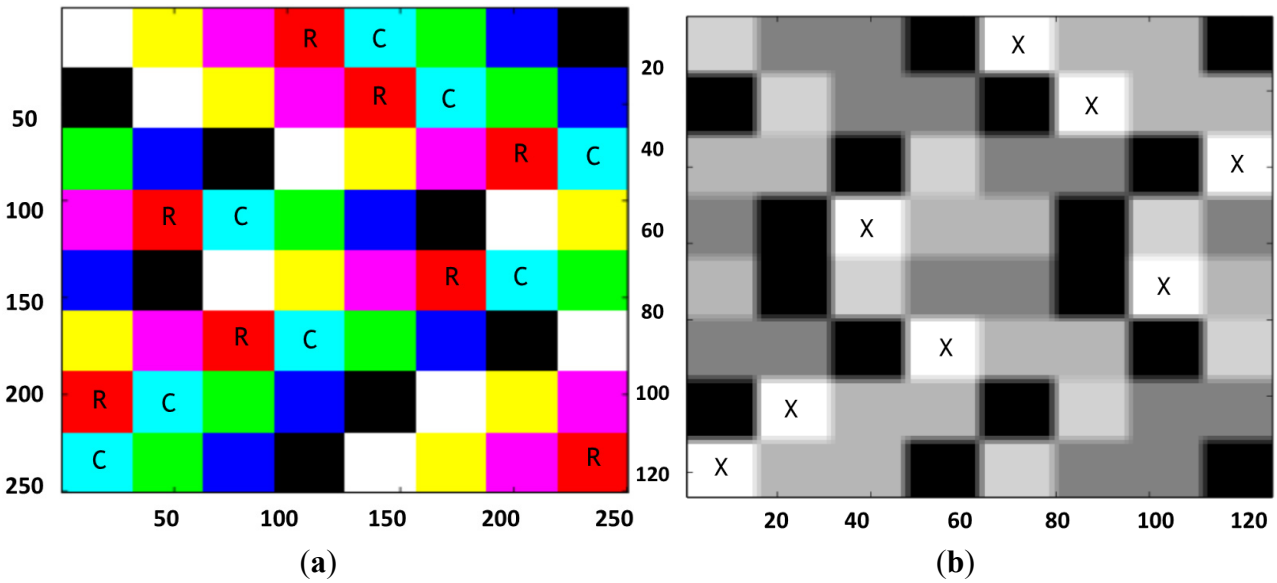

Figure 11. Outputs of the Cyan block before (**a**) and after (**b**) smoothing filter.

As shown in the results of Figure 11b, two sets of masks were chosen for the smoothing operation. The first was set to the color cyan, the color of the block, which is at the lower left corner block of the image (see Figure 11a). The normalized color filter is set for cyan having RGB values set at (0,127,128). The smoothing operation due to convolution of the mask results in Figure 11b (the smaller size 128 × 128 is due to a formatting issue). The second mask for smoothing the red component (255,0,0) results in the image shown in Figure 12.

Table 9. One geometric product for rotor convolution.

Image Size	Full Multivector		ASIC with single core		ASIC with two Core	
Image	Cycles	T (ms)	Cycles	T (ms)	Cycles	T (ms)
128 × 128	1.37×10^6	10.5	1.24×10^6	9.5	6.38×10^5	4.9
256 × 256	5.50×10^6	42.3	4.98×10^6	38.3	2.55×10^6	19.6
512 × 512	2.20×10^7	169.3	1.99×10^7	153.2	1.02×10^7	78.6

As shown in the Tables 9 and 10 the step filtering operation could be performed in 1 GP and 8 GA additions. Table 9 shows the timing and the clock cycles the hardware takes for the GA geometric product and Table 10 for the GA addition operations. The results also show the variation in timing for full multivector color represented as three bivectors and timing within a single core and two cores.

Table 10. Eight geometric additions for rotor convolution.

Image Size	Full Multivector		ASIC with single core		ASIC with two Core	
Image	Cycles	T (ms)	Cycles	T (ms)	Cycles	T (ms)
128 × 128	9.43×10^6	72.5	5.24×10^6	40.3	4.45×10^6	34.2
256 × 256	3.77×10^7	290.3	2.09×10^7	161.3	1.78×10^7	137.1
512 × 512	1.50×10^8	1161	8.38×10^7	645.2	7.13×10^7	548.4

Both the experiments with color sensitive edge detection and smoothing have demonstrated that there is a substantial performance improvement when using the coprocessor. Furthermore, employing simple optimization technique leads to huge benefits. For example, in color sensitive edge detection with an image size 128 × 128, the full multivector expression takes 22.8 ms for 2 GP operations, whereas single core ASIC requires 19.1ms and two cores require 9.8 ms. Similarly for four geometric additions it requires 36.2 ms, 20.1 ms and 17.1 ms, respectively. This leads to 44% (with single core) and 52.7% (with two cores) relative performance improvements. In the case of color smoothing experiment we observe a similar trend with 44% and 52.8% performance improvement with single core and two cores, respectively.

We have shown that with an ASIC implementation it is feasible to achieve data throughput rates, which will provide fast enough operation for video applications, even with this relatively conservative process. For example, taking an image size 128 × 128 we can achieve 46.73 frames per second (fps), 58.48 fps and 29.24 fps for color difference edge detection, color sensitive edge detection and color sensitive smoothing, respectively. The ASIC was implemented as a "proof of concept" and therefore was not optimized for speed, but does demonstrate the advantages of a dedicated co-processor for this type of application, without the compromises inherent in an equivalent FPGA implementation.

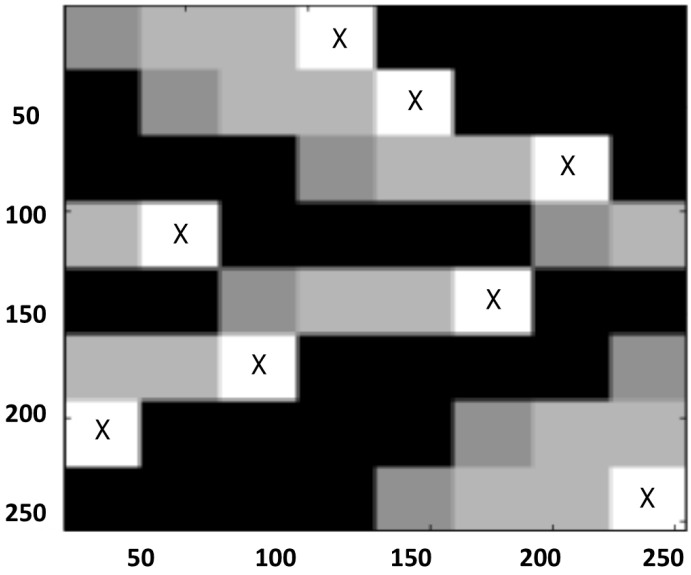

Figure 12. Red only smoothing.

7. Conclusions

This paper presents an overview of the Geometric Algebra fundamentals and convolution operations involving rotors for image processing applications. The discussion shows that the convolution operation with the rotor masks within GA belongs to a class of linear vector filters and can be applied to image or speech signals. Furthermore, it shows that this kind of edge detection is compact and is performed wholly using bivector color images.

The use of the ASIC GA Co-processor for rotor operations was introduced, including a demonstration of its potential for other applications in Vision and Graphics. This hardware architecture is tailored for image processing applications, providing acceptable application performance requirements. The usefulness of the introduced approach was demonstrated by analyzing and implementing three different edge detection algorithms using this hardware. The qualitative analysis for the edge detection algorithm shows the usefulness of GA based computations within image processing applications in general and the potential of the hardware architectures in signal processing applications.

Details are presented of the custom Geometric Algebra Co-Processor that directly targets GA operations and results in significant performance improvement for color edge detection. The contribution of the proposed approach has been demonstrated by analyzing and implementing three different types of edge detection schemes on the GA Co-Processor and FPGA platforms and overall performance gains is reported. A detailed analysis was undertaken which describes not only the raw timings, but also the trade-offs that can be made in terms of resources, area and timing. In addition, the results were also analyzed in the context of larger numbers of operations, where the potential performance benefits can be seen to scale to larger datasets. Future work will be focused on theoretical understanding of this linear filter and the frequency response, which would be useful in developing further algorithms.

Acknowledgments

Biswajit Mishra would like to thank the support of University of Southampton for his research studies in the form of a PhD studentship and scholarship.

Author Contributions

Biswajit Mishra developed the proposal of the study, implemented the design in hardware, built the test setup, carried out the analysis of results and took care of most of the writing; Peter Wilson guided the entire process as well as the analysis of the proposal and made corrections; Reuben Wilcock reviewed the experimental and simulation results and wrote several sections the manuscript; All authors contributed to the analysis of the simulation results, in order to provide a more comprehensible final version.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Koschan, A.; Abidi, M. Detection and classification of edges in color images. *IEEE Signal Proc. Mag.* **2005**, *22*, doi:10.1109/MSP.2005.1407716.
2. Lukac, R.; Smolka, B.; Martin, K.; Plataniotis, K.N.; Venetsanopoulos, A.N. Vector filtering for colour imaging. *IEEE Signal Proc. Mag.* **2005**, *22*, 74–86.
3. Trussel, H.J.; Saber, E.; Vrhel, M. Color image processing. *IEEE Signal Proc. Mag.* **2005**, *22*, doi:10.1109/MSP.2005.1407711.
4. Astola, J.; Haavisto, P.; Neuovo, Y. Vector median filters. *Proc. IEEE* **1990**, *78*, doi:10.1109/5.54807.
5. Trahanias, P.E.; Venetsanopoulos, A.N. Color edge detection using vector order statistics. *IEEE Trans. Image Proc.* **1993**, *2*, doi:10.1109/83.217230.
6. Machuca, R.; Phillips, K. Applications of vector fields to image processing. *IEEE Trans. Pattern Anal. Mach. Intell.* **1983**, *5*, 316–329.
7. Sangwine, S.J. Fourier transforms of colour images using quaternion or hypercomplex numbers. *IEE Electron. Lett.* **1996**, *32*, 1979–1980.
8. Bayro-Corrochano, E.; Flores, S. Color edge detection using rotors. In *Applications of Geometric Algebras in Computer Science and Engineering*; Dorst, C.D.L., Lasenby, J., Eds.; Birkhauser: Boston, MA, USA, 2002.
9. Mishra, B.; Wilson, P. Color edge detection hardware based on geometric algebra. In *3rd European Conference on Visual Media Production CVMP 2006*; IET: London, UK, 2006; pp. 115–121.
10. Ebling, J. Clifford fourier transform on vector fields. *IEEE Trans. Visual. Comput. Graph.* **2005**, *11*, 469–479.
11. Tabard, T.; Saint-Jean, C.; Berthier, M. A metric approach to *nD* images edge detection with clifford algebras. *J. Math. Imaging Vis.* **2009**, *33*, 296–312.
12. Nadernejad, E. Edge detection techniques: Evaluations and comparisons. *Appl. Math. Sci.* **2008**, *2*, 1507–1520.
13. Zhou, C.; Mel, B.W. Cue combination and color edge detection in natural scenes. *J. Vis.* **2008**, *8*, doi:10.1167/8.4.4.

14. Burgeth, B.; Kleefeld, A. An approach to color-morphology based on Einstein addition and Loewner order. *Pattern Recog. Lett.* **2014**, *47*, 29–39.
15. Van de Gronde, J.J.; Roerdink, J.B.T.M. Group-invariant colour morphology based on frames. *IEEE Trans. Image Proc.* **2014**, *23*, 1276–1288.
16. Evans, A.N.; Liu, X.U. A morphological gradient approach to color edge detection. *IEEE Trans. Image Proc.* **2006**, *15*, 1454–1463.
17. Hestenes, D. *New Foundations for Classical Mechanics*; D. Reidel Publishing Company: Dordrecht, The Netherlands, 1986.
18. Dorst, L.; Mann, S. Geometric algebra: A computational framework for geometrical applications (I). *IEEE Comput. Graph. Appl.* **2002**, *22*, 24–31.
19. Goldman, R. Computer graphics in its fifth decade: Ferment at the foundations. In Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, Washington, DC, USA, 2003.
20. Lasenby, J.; Fitzgerald, W.J.; Lasenby, A.N.; Doran, C.J.L. New geometric methods for computer vision: An application to structure and motion estimation. *Int. J. Comput. Vision* **1998**, *26*, 191–213.
21. Mishra, B.; Wilson, P. VLSI implementation of a geometric algebra micro architecture. In Proceedings of the International Conference on Clifford Algebras and their Applications, Toulouse, France, 21–29 May 2005.
22. Mishra, B.; Wilson, P. Hardware implementation of a geometric algebra processor core. In Proceedings of the International Association for Mathematics and Computers in Simulation, International Conference on Advancement of Computer Algebra, Nara, Japan, 31 July–3 August 2005.
23. Mishra, B.; Kochery, M.P.; Wilson, P.; Wilcock, R. A novel signal processing coprocessor for *n*-dimensional geometric algebra applications. *Circuit. Syst.* **2014**, *5*, 274–291.
24. IEEE Computer Society Standards Committee. Working group of the Microprocessor Standards Subcommittee and American National Standards Institute. *IEEE Standard for Binary Floating-Point Arithmetic*; IEEE Computer Society Press: Silver Spring, MD, USA, 1985.
25. Perwass, C.; Gebken, C.; Sommer, G. Implementation of a clifford algebra co-processor design on a field programmable gate array. In *CLIFFORD ALGEBRAS*: *Application to Mathematics, Physics, and Engineering, ser. Progress in Mathematical Physics*; Ablamowicz, R., Ed.; Birkhauser: Boston, MA, USA, 2003; pp. 561–575.
26. Gentile, A.; Segreto, S.; Sorbello, F.; Vassallo, G.; Vitabile, S.; Vullo, V. CliffoSor: A parallel embedded architecture for geometric algebra and computer graphics. In Proceeding of the Seventh International Workshop on Computer Architecture for Machine Perception, Palermo, Italy, 4–6 July 2005; pp. 90–95.
27. Lange, H.; Stock, F.; Koch, H.D. Acceleration and energy efficiency of a geometric algebra computation using reconfigurable computers and GPUs. In Proceedings of the 17th IEEE Symposium on Field Programmable Custom Computing Machines, Napa, CA, USA, 5–7 April 2009.
28. Stock, F.; Hildenbrand, D. FPGA-accelerated color edge detection using a geometric-algebra-to-verilog compiler. In Proceedings of the 2013 International Symposium on System on Chip, Tampere, Finland, 23–24 October 2013.

29. Franchini, S.; Gentile, A.; Sorbello, F.; Vassallo, G.; Vitabile, S. An embedded, FPGA-based computer graphics coprocessor with native geometric algebra support. *VLSI J. Integr.* **2009**, *42*, 346–355.
30. Franchini, S.; Gentile, A.; Sorbello, F.; Vassallo, G.; Vitabile, S. Design and implementation of an embedded coprocessor with native support for 5D, quadruple-based clifford algebra. *IEEE Trans. Comput.* **2013**, *62*, 2366–2381.
31. Dorst, L.; Mann, S.; Bouma, T.A. GABLE: A geometric algebra learning environment. Available online: https://staff.fnwi.uva.nl/l.dorst/GABLE/ (accessed on 10 October 2011).
32. Fontijne, D. Gaigen Homepage. Available online: http://www.sourceforge.net/projects/gaigen (accessed on 10 October 2011).
33. Sangwine, S.J. Colour image edge detector based on quaternion convolution. *IEE Electron. Lett.* **1998**, *34*, 969–971.
34. Evans, C.J.; Sangwine, S.J.; Ell, T.A. Colour-sensitive edge detection using hypercomplex filters. In Proceedings of the EUSIPCO 2000, 10th European Signal Processing Conference, Tampere, Finland, 5–8 September 2000; pp. 107–110.
35. Evans, C.J.; Sangwine, S.J.; Ell, T.A. Hypercomplex color-sensitive smoothing filters. In Proceeding of the 2000 International Conference on Image Processing, Vancouver, BC, Canada, 10–13 September 2000.